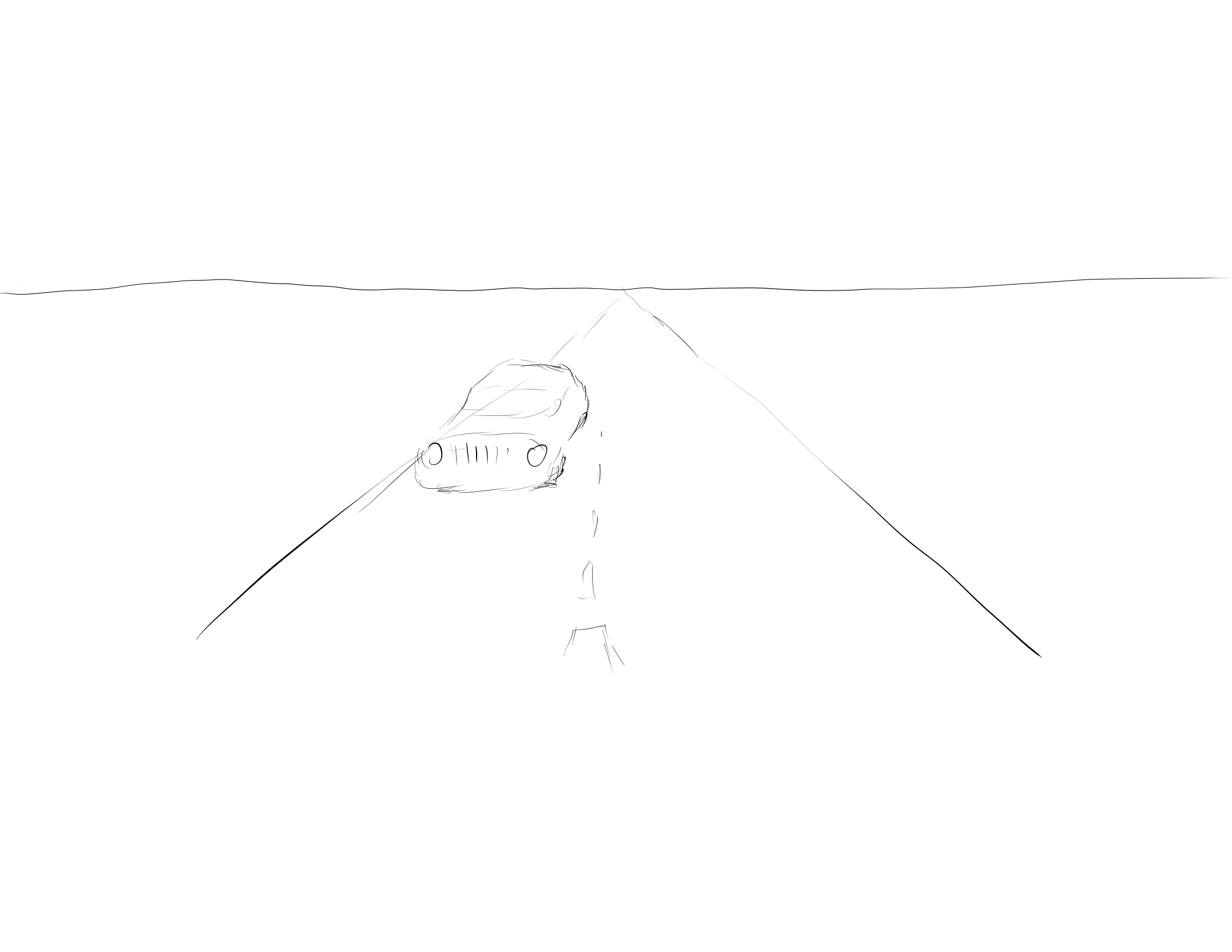
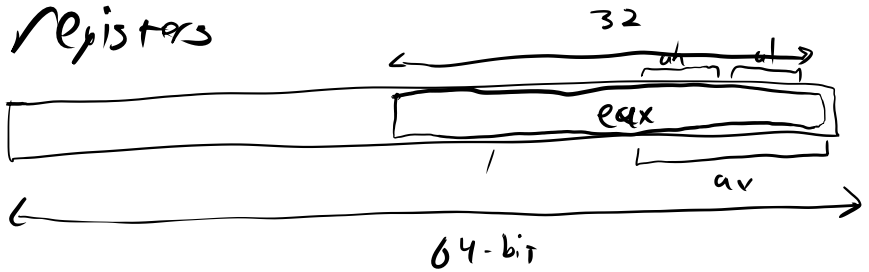


mov
add

Registers



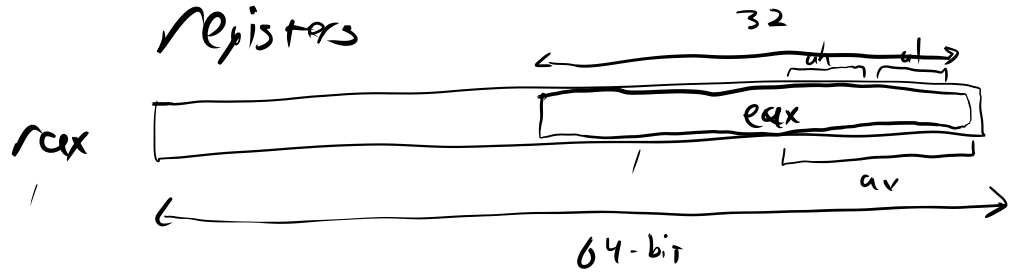
movq – 64 bit

movl – 32 bit

=

0	$r_0 \rightarrow r_1$	movq %rax, %rcx
3	$r_0 \rightarrow M(r_1)$	movq %rax, (%rcx)
4	$M[r_1] \rightarrow r_0$	movq (%rcx), %rax
6.0	$21 \rightarrow r_0$	movq $21, %rcx

mov
add



movq – 64 bit

movl – 32 bit

+=

0	r0 → r1	addq %rax, %rcx
3	r0 → M[r1]	addq %rax, (%rcx)
4	M[r1] → r0	addq (%rcx), %rax
6.0	21 → r0	addq $21, %rcx

Jumps

Unconditional

jmp foo

foo:

Conditional

jl	<	
jle	<=	
je	==	
jne	!=	
jg	>	
jge	>=	
ja	>	unsigned
jb	<	unsigned
js	sign bit set	
jo		

Condition Codes (flags)

even - math op
- special cmp test

set CC

addq $-5, %rax

:

je foo

cmpq %rax, %rdx — set CC as rdx - rax <=> 0

jle foo

testq %rax, %rdx — set CC as rax & rdx

```
while (x < 10)
    x += 1
```

```
top:    x - 10 >= 0
    if (x >= 10) goto end
    x += 1
    goto top
end:
```

x = %rax

```
Top:
        cmpq    $10, %rax       // rax - 10
        Jge     end
        addq    $1, %rax
        Jmp     top
end:
```

Functions

args: %rdi, %rsi, ...

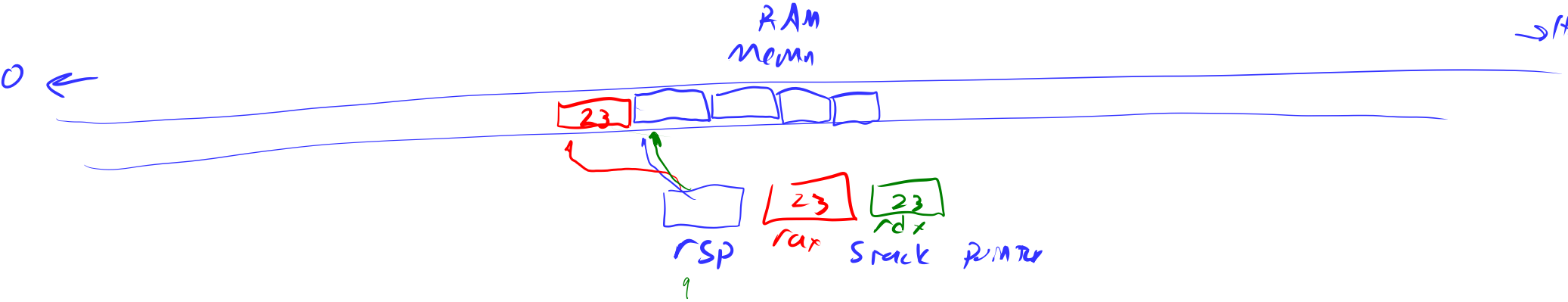
int f(int x, int y) : ⟶ label ⟶ f:

=

return 3 ⟶ movq $3, %rax
retq ⟶ %rax

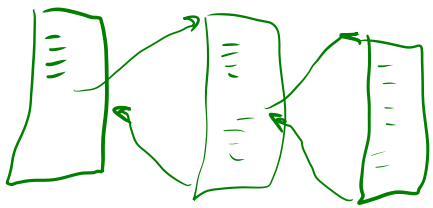
f(3,4) ⟶ movq $3, %rdi
movq $4, %rsi
callq f

The Stack



pushq %rax

popq %rdx



call - push return address

ret - pop an address to jump to